WHAT IS CLAIMED IS:

1. In a scheduler for computer code wherein certain operations are likely to stall execution of the computer code and thereby provide latency for completion of one or more pre-executable operations, a method of scheduling certain of the operations, the method comprising:

for one or more sequences of operations that follow a speculation boundary and that define respective dependency chains, including pre-executable operations, which lead to likely stalls, representing speculative copies thereof as duplicate chains; and

scheduling operations of the computer code, wherein the scheduling of operations from the duplicate chains is performed without regard to dependence of respective original operations on the speculation boundary, thereby scheduling certain of the operations above the speculation boundary into position preceding at least one of the operations likely to stall execution of the computer code.

2. A method, as recited in claim 1,

wherein the likely stalls include likely cache misses.

3. A method, as recited in claim 1,

wherein the dependency chains include address chains leading to memory access operations likely to miss in a cache.

4. A method, as recited in claim 1,

wherein the pre-executable operations include prefetch instructions.

5. A method, as recited in claim 1,

wherein the pre-executable operations include speculative operations.

6. A method, as recited in claim 1,

wherein the operations likely to stall execution include memory access instructions.

7. A method, as recited in claim 1, wherein the operations likely to stall execution include operations selected from the set of:

a load operation;

first use of a load operation;

a store operation;

a branch operation;

a multi-cycle computational operation;

an iterative or recursive operation;

a communications operation;

an input/output (I/O) operation;

a synchronization operation; and

a co-processor operation.

8. A method, as recited in claim 1, wherein the speculation boundary is defined by one of:

a store operation;

a branch operation;

a join operation;

an iterative or recursive operation;

a communications operation;

an input/output (I/O) operation;

a synchronization operation; and

a co-processor operation.

9. A method, as recited in claim 1, further comprising:

inserting the pre-executable operations into the computer code.

10. A method, as recited in claim 1, further comprising:

profiling the computer code to identify the likely stalls.

11. A method, as recited in claim 1, further comprising:

upon reaching the speculation boundary, deleting unscheduled operations of the duplicate chains and continuing to schedule respective original operations.

12. A method, as recited in claim 1, further comprising:

deleting from the original operations, pre-executable operations for which a respective speculative copy is scheduled.

13. A method of hiding latency in computer code wherein certain operations thereof are likely to stall execution, the method comprising:

identifying sequences of operations that define respective original dependency chains that lead to likely stalls and for at least some of the identified sequences, representing duplicate dependency chains thereof; and

scheduling at least some operations from the duplicate dependency chains above at least one of the likely-to-stall operations.

14. The method of claim 13,

wherein the operations scheduled from the duplicate dependency chains are scheduled above a speculation boundary upon which the original dependency chains depend.

15. The method of claim 13,

wherein dependency chains are defined at least in part by address dependencies.

16. The method of claim 13, further comprising:

upon reaching a corresponding speculation boundary, removing otherwise unscheduled operations of the duplicate dependency chains.

17. The method of claim 13, wherein the operations likely to stall execution include operations selected from the set of:

a load operation;

first use of a load operation;

a store operation;

a branch operation;
a multi-cycle computational operation;
an iterative or recursive operation;
a communications operation;
an input/output (I/O) operation;
a synchronization operation; and
a co-processor operation.

18. The method of claim 14, wherein the speculation boundary is defined by one of:

a store operation;
a branch operation;
a join operation;
an iterative or recursive operation;
a communications operation;
an input/output (I/O) operation;
a synchronization operation; and
a co-processor operation.

19. The method of claim 14, wherein the speculation boundary is defined by an operation that has irreversible side-effects.

20. The method of claim 13,
wherein the operations likely to stall execution include memory access operations.

21. The method of claim 13, further comprising:
for at least load-type ones of the operations, inserting corresponding prefetch operations.

22. The method of claim 13, further comprising:
converting load-type ones of the scheduled operations to speculative counterpart operations.

23. The method of claim 13, further comprising:
converting load-type ones of the scheduled operations to non-faulting loads.

24. The method of claim 13, further comprising
responsive to the scheduling of a prefetch operation from one of the duplicate dependency chains, disposing of a corresponding prefetch operation from a corresponding one of the original dependency chains.

25. The method of claim 13, further comprising:
selecting for the scheduling, particular ones of the operations from the duplicate dependency chains based at least in part on chain length.

26. The method of claim 13,
wherein the likely to stall operations include memory operations predicted to miss in a cache.

27. The method of claim 13,
wherein the likely to stall operations include store-type operations predicted to miss in a cache.

28. The method of claim 13,
wherein the likely to stall operations include operations that stall an execution pipeline.

29. The method of claim 13,
wherein the dependency chains include load-type and prefetch operations.

30. The method of claim 13,
wherein the dependency chains include operations other than load-type and prefetch operations.

31. The method of claim 13,
wherein the dependency chains include operations involved in address calculations.

32. The method of claim 13,

wherein the duplicate dependency chains are represented as copies of the respective original dependency chains with speculation boundary dependencies removed or ignored.

33. The method of claim 13,

wherein the dependency chains are represented a directed acyclic graph of dependencies amongst the corresponding operations.

34. The method of claim 33, wherein the dependencies include one or more of:

register dependencies;

branch dependencies; and

memory dependencies.

35. The method of claim 13, realized in an optimizing compiler.

36. The method of claim 13, realized in a just-in-time (JIT) compiler.

37. A method of making a computer program product that encodes program code for which memory access latency is at least partially hidden on execution thereof, the method comprising:

for operations that form addressing chains that lead to a likely cache miss, representing speculative copies thereof; and

scheduling the speculative copies without regard to a corresponding speculation boundary, wherein operations of the speculative copies are scheduled above the corresponding speculation boundary and above a preceding operation that is likely to stall.

38. The method of claim 37,

encoding the scheduled operations as part of the program code.

39. The method of claim 37,

wherein preceding operation that is likely to stall is a likely cache miss.

40. A method of scheduling code comprising:
inserting prefetch operations into code;
identifying those operations likely to stall execution of the code; and
scheduling some of the prefetch operations above respective likely-to-stall operations thereby exploiting latency for completion of the scheduled prefetch operations.

41. The method of claim 40,
wherein the operations likely to stall execution include memory access instructions likely to miss in a cache.

42. A computer program product encoded in one or more computer readable media, the computer program product comprising:
an execution sequence of instructions,
the execution sequence including subsequence that includes a speculative load instruction that feeds a subsequent prefetch instruction.

43. The computer program product of claim 42, further comprising:
one or more instructions disposed between the speculative load instruction and the subsequent prefetch instruction in the execution sequence.

44. The computer program product of claim 42, further comprising:
a martyr instruction that follows the speculative load instruction and the prefetch instruction which, upon execution, provides at least a portion of a latency therefor.

45. The computer program product of claim 42,
prepared by a program scheduler that inserts prefetch instructions into the execution sequence and schedules speculative duplicates of at least some load instructions together with corresponding prefetch instructions above speculative boundaries therein.

46. The computer program product of claim 42,

wherein the one or more computer readable media are selected from the set of a disk, tape or other magnetic, optical, semiconductor or electronic storage medium and a network, wireline, wireless or other communications medium.

47. An apparatus comprising:

a code preparation facility for transforming schedulable code into scheduled code; and

means for scheduling speculative copies of operations that form dependency chains that lead to a likely stall, the scheduling placing the speculative operations above a preceding at least one other operation that is itself likely to stall, thereby hiding in the scheduled code latency of the speculative operations.

48. The apparatus of claim 47, further comprising:

means for inserting pre-executable operations into the schedulable code, wherein at least some of the pre-executable operations are scheduled be the scheduling means as the speculative operations for which latency is hidden.

49. The apparatus of claim 47, further comprising:

means for identifying likely-to-stall operations of schedulable code.